High-Multiplicity Scheduling on One Machine with Forbidden Start and Completion Times

Michaël Gabay, Christophe Rapine, Nadia Brauner

Noname manuscript No.
(will be inserted by the editor)

High-Multiplicity Scheduling on One Machine with Forbidden Start and Completion Times

Michaël Gabay · Christophe Rapine · Nadia Brauner

Received: date / Accepted: date

Abstract We are interested in a single machine scheduling problem where jobs can neither start nor end on some specified instants, and the aim is to minimize the makespan. This problem may model the situation where an additional resource, subject to unavailability constraints, is required to start and to finish a job. We consider in this paper the High-Multiplicity version of the problem, when the input is given using a compact encoding. We present a polynomial time algorithm for large diversity instances (when the number of different processing times is greater than the number of forbidden instants). We also show that this problem is Fixed-Parameter Tractable when the number of forbidden instants is fixed, regardless of jobs characteristics.

Keywords Scheduling · High-Multiplicity · Availability Constraints · Parametrized Complexity

1 Introduction

We consider a scheduling problem on one machine where a set of instants is given, such that no job is allowed to start or to complete at any of these instants. We refer to such an instant as a *forbidden start & end instant* (Fse). Forbidden instants may arise when jobs need some additional resources at launch and completion and these resources are not continuously available. This may be the case if the additional resources are shared with other activities. For example, consider the situation where the jobs are processed by an automated device during a specified amount of time, but a qualified operator is required on setup and completion. While the device is continuously available, the operators have days off and other planed activities. On these days, jobs can be performed by the device, but none can start or complete. We encountered this problem in chemical industry through a collaboration with the *Institut Français du Pétrole*. In their problem, jobs were chemical experiments whose durations typically last between 3 days and 3 weeks. A chemist is required on jobs start and completion to control the process. Each intervention of the chemist can be performed within an hour, but requires of course a chemist to be available and present in the laboratory. For more details on this application, we refer the reader to Brauner et al (2009) and Rapine et al (2012).

Notice that, contrary to a classical unavailability constraint, the machine can be processing a job during an Fse instant, as long as it started its execution before the forbidden instant and will complete after it. We restrict to integer values for the data and to schedules where all the jobs start and complete at integer instants. The objective is to minimize the makespan $C_{\max}$. Using Graham notations, the problem is denoted by $1|\text{Fse}|C_{\max}$. As an example, consider the instance where instants 3, 4, 6 and 9 are Fse instants and 5 jobs have to be scheduled: a and b of duration 1, c and d of duration 2 and e of duration 4. On Figure 1 and 2, forbidden instants are represented on the time axis by dashed rectangles. The sequence of jobs (a, e, c, b, d) leads to an idle-free schedule represented Figure 1 ; the

M. Gabay
Grenoble-INP / UJF-Grenoble 1 / CNRS, G-SCOP UMR5272 Grenoble, F-38031, France
E-mail: michael.gabay@g-scop.grenoble-inp.fr

C. Rapine
Université de Lorraine, Laboratoire LGIPM, Ile du Saulcy, Metz, F-57045, France
E-mail: christophe.rapine@univ-lorraine.fr

N. Brauner
Grenoble-INP / UJF-Grenoble 1 / CNRS, G-SCOP UMR5272 Grenoble, F-38031, France
E-mail: nadia.brauner@g-scop.grenoble-inp.fr

makespan of this schedule is 10. On Figure 2, we have represented the scheduling of the jobs according to the LPT sequence (e, d, c, b, a), that is in non-increasing order of the processing times. In order to respect the forbidden instants, two idle slots are used in the schedule. One can check that the SPT sequence (a, b, c, d, e) leads to a worse schedule, of makepsan 14.

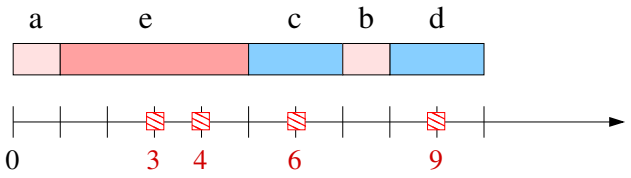

Fig. 1: Sequence (a, e, c, b, d). The schedule is idle-free and completes at time 10.

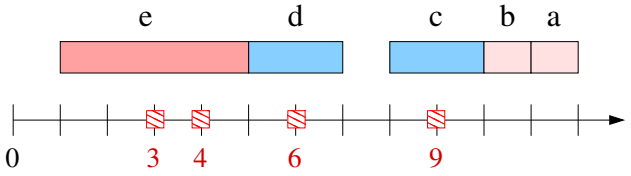

Fig. 2: Sequence (e, d, c, b, a). The schedule completes at time 12.

The problem of scheduling jobs on a single machine where a set of time slots is forbidden for starting or completing the jobs has been first investigated by Billaut and Sourd (2009). They considered the case where some time slots are forbidden for starting the jobs, namely the Fs instants (for *forbidden start*). They proved that minimizing the makespan is polynomially solvable if the number of forbidden start times is fixed, and $\mathcal{NP}$-hard in the strong sense if this number is part of the input. Their algorithm runs in time $\mathcal{O}(n^{2k^2+2k-1})$, where k denotes the number of Fs instants and n is the number of jobs. They also established that if there are at least $2k(k+1)$ distinct processing times of the jobs in the instance, then an idle-free schedule exists. Rapine and Brauner (2013) generalized this results: they established that having $k+1$ distinct processing times is a sufficient condition to ensure the existence of an idle-free schedule in presence of k Fse instants. Such an optimal schedule can be found in $O(k^3 n)$. As a consequence, the overall complexity to solve the problem for a fixed number of forbidden instants is reduced to $O(n^k)$. Chen et al (2013) Consider the same problem with a different objective function, namely the total completion time.

High-multiplicity encoding

The number of *types* of jobs, that is the number of different job durations, play a central role in the above mentioned results. Hence, it is natural to consider a compact encoding where similar jobs are grouped together. The problem then falls in the field of High-Multiplicity Scheduling introduced by Hochbaum and Shamir (1991). Compared to a traditional encoding, where each job is described, in a high-multiplicity (HM) encoding, each *type* is described only once, along with its multiplicity (the number of jobs of this type). Thus the size of a HM encoding depends linearly on the number of types but only logarithmically on the number of jobs. As a consequence, a polynomial time algorithm under the standard encoding may become exponential under a HM encoding of the input, which is the case of the previously mentioned algorithms. HM scheduling and more generally HM combinatorial optimization has become an active domain in recent years, see (Brauner et al 2005; Clifford and Posner 2001; Filippi and Agnetis 2005; Filippi and Romanin-Jacur 2009).

The goal of this paper is to explore the complexity of problem $1|\text{Fse}|C_{\max}$ under a high-multiplicity encoding of the input. We show that essentially the main results established in the literature under a standard encoding remain valid under a HM encoding. Specifically, we propose in Section 2 a polynomial time algorithm for large diversity instances, that is when the number of types is greater than the number of Fse instants. In Section 3, we also prove that the general problem remains polynomial when k is fixed. We first introduce the following notations which will be used in the remaining of the paper.

Notations

Throughout the paper k denotes the number of Fse instants in the instance. Let γ_i be the i-th Fse instant with $\gamma_1 < \gamma_2 ... < \gamma_k$. We denote by $\mathcal{F} = \{\gamma_1, \ldots, \gamma_k\}$ the set of the Fse instants. Two jobs are of different types if and only if their processing times are different. The number of types of jobs in the instance is denoted by s. Without loss of generality, we index the types by decreasing order of the processing times of their jobs. The set of jobs to schedule is represented in a HM encoding by a multiplicity vector $(m_1, \ldots, m_s)$, together with a processing times vector $(p_1, \ldots, p_s)$, where m_i and p_i are respectively the number of jobs of the ith type and its corresponding processing time. The number of jobs is $n = \sum_{i=1}^{s} m_i$. The instance of Figures 1 is thus represented by the processing times vector $(4, 2, 1)$ and the multiplicity vector $(1, 2, 2)$. A job is said to

cross an FSE instant γ_i if it starts its processing before γ_i and ends after γ_i. For instance in Figure 1, the job e crosses the first two FSE instants.

We denote by $|x|$ the size of the input under a HM encoding. We have $|x| = \mathcal{O}(s \log n + s \log p_1 + k \log \gamma_k)$. Hence $|x|$ can be in $\mathcal{O}(s \log n)$ while the algorithm proposed in Rapine and Brauner (2013) runs in time $\mathcal{O}(k^3 n)$, which can be exponential with respect to $|x|$.

In HM scheduling, it may not be obvious to determine whether or not schedules can be described with a compact encoding, *i.e.* polynomial in $|x|$. For the problem we consider, it is readily that the schedule of the jobs between two forbidden instants is meaningless (provided unnecessary idle-times are not inserted). As a consequence any schedule has a polynomial encoding as a sequence of k vectors $(m_i^1, \ldots, m_i^s)$ and k pairs (j_i, s_i), where m_i^j is the number of jobs of type j scheduled between γ_{i-1} and γ_i ; j_i is the job crossing γ_i and s_i its starting time.

An instance is denoted by $x = (N, \mathcal{F})$ where N is the set of jobs. We say that an instance is of *large diversity* if $s > k$, that is, if the number of distinct types is greater than the number of forbidden instants. In the reverse situation, we say that the instance is of *small diversity*.

2 A polynomial time algorithm for large diversity instances

In this section, we design a polynomial algorithm for large diversity instances. Rapine and Brauner (2013) proved that, in such cases, there exists and idle-free schedule:

Theorem 1 (Rapine and Brauner (2013)) *If $s > k$ and 0, $p(N) \notin \mathcal{F}$, then there exists a feasible schedule without idle time.*

They also presented an algorithm, called L-partition, finding an idle-free schedule for large diversity instances in $\mathcal{O}(k^3 n)$ time, where $n = \sum_{i=1}^{s} m_i$ is the number of jobs. Although linear in the number of jobs, this algorithm is not polynomial with a high-multiplicity encoding, except if the multiplicity of each type is bounded by a constant. In particular if only one job is associated with each type, the L-partition algorithm runs in time $\mathcal{O}(k^3 s)$. We use this fact in our approach.

To design a polynomial time algorithm under a HM encoding, we need to schedule more than one job at a time. We also need an efficient way to decompose the problem. Consider a large diversity instance $x = (N, \mathcal{F})$. Notice that Theorem 1 ensures that an optimal schedule is idle free, assuming that neither instant 0 nor instant $p(N)$ is forbidden. A schedule is said *partial* if only a subset of the jobs is scheduled. We introduce the following definition:

Definition 1 A partial schedule π is an *optimal prefix* if there exists an optimal schedule of the form $\pi\sigma$.

Consider a partial schedule π completing at time t. Looking at the definition, deciding if π is an optimal prefix may request to compute an optimal schedule for the whole instance. However, by Theorem 1, a sufficient condition for π to be an optimal prefix is that π is idle-free, and that the remaining instance $x' = (N', \mathcal{F}' = \mathcal{F} \cap [t, +\infty[)$ is a large diversity instance. Indeed, it guarantees the existence of an idle-free schedule σ for the remaining jobs to schedule after time t.

If we are able to find an optimal prefix π, the problem is reduced to finding an optimal schedule starting at time t on the remaining set N' of jobs. We can then look again for an optimal prefix π' on the remaining large diversity instance x'. However, for this decomposition to be efficient, we need to bound the number of times an optimal prefix is searched for. We say that a prefix π is *efficient* if it is optimal and crosses at least one forbidden instant. It is then immediate that at most k efficient prefixes need to be computed to build an optimal schedule.

Algorithm 1 Optimal Prefix Algorithm

Require: a large diversity instance $(N,\mathcal{F})$ with types indexed by decreasing order of processing times p_j.
Ensure: an optimal prefix π
  set $m_i = m_i - 1$ for $i = 1$ to $k+1$
  $i = 1$; $t = 0$; $\pi = \emptyset$;
  while $i \leq s$ and $t + m_i p_i < \gamma_1$ **do**
    {Append the m_i jobs of type i to π}
    $\pi = \pi(i, m_i)$; $t = t + m_i p_i$; $i = i + 1$;
  end while
  if $i > s$ **then**
    return π {Only $k+1$ jobs remain to schedule}
  end if
  {Append as many jobs of type i as possible, before γ_1}
  $\alpha = \lceil (\gamma_1 - t)/p_i \rceil - 1$; $\pi = \pi(i, \alpha)$; $t = t + \alpha p_i$;
  {Extend π to complete after time γ_1}
  for $l = 1$ to $k+1$ such that $t + p_l \geq \gamma_1$ **do**
    if $t + p_l \notin \mathcal{F}$ **then**
      return $\pi(l, 1)$
    end if
  end for
  for $l = 2$ to $k+1$ such that $t + p_l < \gamma_1$ **do**
    if $t + p_l + p_1 \notin \mathcal{F}$ **then**
      return $\pi(l, 1)(1, 1)$
    end if
  end for

Algorithm 1 finds an (efficient) optimal prefix. The main idea of the algorithm is to put aside initially one job of each of the $k+1$ largest types. Let B be this set of jobs. This reserve B is used to ensure that the

remaining instance is of large diversity. Notice that we can afford to use one of these jobs each time a forbidden instant is crossed. We call *additional* jobs the set $A = N \backslash B$. The algorithm iteratively schedules all the additional jobs of type 1, then all the additional jobs of type 2, and so on. Recall that types are indexed in decreasing order of the processing times, thus we simply follow a LPT sequence for the additional jobs. We keep scheduling additional jobs as long as they fit before the first forbidden instant γ_1. When this process halts on some index i, either only the jobs from the set B remain to schedule, or there is not enough room left before γ_1 to schedule all the additional jobs of the ith type. In the latter case, the algorithm schedules as much jobs of type i as possible before γ_1. Then, it tries to cross the forbidden instant γ_1. In order to keep a large diversity instance, we ensure that each job of B scheduled allows to cross at least one forbidden instant. This way the algorithm outputs an efficient prefix. In the other case, all additional jobs have been scheduled and the partial schedule returned is optimal but not efficient, since the first FSE instant is not crossed. However, we are in the situation where the remaining large diversity instance contains only one job per type, and we have exactly $k+1$ types. We can use the L-partition algorithm to solve it efficiently, in time $\mathcal{O}(k^4)$. The correctness of the algorithm is summarized in the following lemma:

Lemma 1 *Given a large diversity instance $x = (N, \mathcal{F})$, Algorithm 1 delivers an optimal prefix π. In addition, if $x' = (N', \mathcal{F}')$ is the remaining instance to schedule, then x' is a large diversity instance and:*

1. *either $|\mathcal{F}'| < |\mathcal{F}|$, that is π is an efficient prefix,*
2. *or $|N'| = |\mathcal{F}| + 1$ and all the remaining jobs have distinct processing times.*

Proof Let $(N', \mathcal{F}')$ be the instance remaining to schedule at the end of Algorithm 1. Recall that B denotes a set with exactly one job of the $k+1$ largest types of N and $A = N \backslash B$ is the set of the additional jobs. If only the set B remains to schedule at the end of the algorithm, then we are clearly in the second case of our claim: $|N'| = k+1$. Otherwise the algorithm has stopped the first loop on a type i such that all the additional jobs cannot be scheduled before γ_1. At this point, there is at least one unscheduled job of type i remaining in A, and possibly another in B, if $i \leq k+1$. Let $t < \gamma_1$ be the current completion time of the schedule, and consider the partition $B = \mathcal{S} \cup \mathcal{L}$ defined by $\mathcal{L} = \{j \in B \mid t + p_j \geq \gamma_1\}$ and $\mathcal{S} = B \backslash \mathcal{L}$. Notice that $\mathcal{L}$ is not empty as $t + p_i \geq \gamma_1$; in particular a job of type 1 belongs to $\mathcal{L}$. By construction the prefix algorithm tries to extend π in order to complete after the first forbidden instant γ_1. We have to prove that it will always succeed, and that $(N', \mathcal{F}')$ is a large diversity instance. We denote by s' the number of distinct types of jobs in the remaining instance x' and by $k' = |\mathcal{F}'|$ the number of FSE instants appearing after time t.

In the following, we show that if π completes after the lth forbidden instant, at most l jobs of B have been scheduled in π. As a consequence, $s' \geq |B| - l > k - l \geq k'$ and $(N', \mathcal{F}')$ is a large diversity instance. Consider the last two loops of the algorithm. If one job of $\mathcal{L}$ can be scheduled, the property clearly holds as π completes after time γ_1. If there is no such job, then for all jobs j of $\mathcal{L}$, $t + p_j$ is a forbidden instant while any job of $\mathcal{S}$ can be scheduled before time γ_1. Therefore a simple counting argument, illustrated Figure 3, ensures that there exists a job $s \in \mathcal{S}$ which can be scheduled at time t immediately followed by a job of type 1. If $t + p_s + p_1 \geq \gamma_2$, *i.e.* π completes after time γ_2, we are done. Otherwise, we have $t + p_1 < \gamma_2$. In this case $k' = k - 1$, while we apparently use 2 jobs of B. However, instant $t + p_1$ is forbidden ; in fact we have $t + p_1 = \gamma_1$ and as a consequence $i = 1$. As we noticed, there is at least one unscheduled job of type i in A. Since $i = 1$, we can use an additional job of type 1, instead of using a job of type 1 from B. We have $s' \geq s - 1$ which completes the proof. □

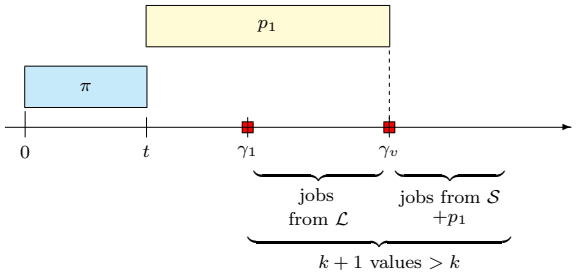

Fig. 3: Counting argument: the first FSE instants can be crossed using one or two jobs from B.

In order to deliver an optimal schedule, we iteratively call the prefix algorithm on the remaining instance as long as we obtain an efficient prefix. Otherwise, we are in the second case of Lemma 1, which corresponds to the basis of the recursion: we simply solve instance $x' = (N', \mathcal{F}')$ using the L-partition algorithm. Since N' contains at most $(k+1)$ jobs, the running time of the L-partition algorithm on this instance is in $\mathcal{O}(k^4)$. We have the following theorem:

Theorem 2 *Problem $1|\text{FSE}|C_{\max}$ is polynomial under HM encoding for large diversity instances, and can be solved in time $\mathcal{O}(sk + k^4)$*

Proof From the above discussion, we only need to establish the time complexity of the algorithm, its correctness being a direct consequence of Lemma 1. We use

the classic convention that basic operations on integers (addition, division...) are performed in constant time. Then the time complexity of Algorithm 1 is in $\mathcal{O}(k+s)$, which is in $\mathcal{O}(s)$ for large diversity instances. To solve Problem $1|\text{FSE}|C_{\max}$, we call Algorithm 1 on the set of unscheduled jobs as long as there is still some forbidden instants in the future or that this set is not reduced to B. Thus we have at most k calls to Algorithm 1, possibly followed by a call to L-partition algorithm on an instance containing at most $k+1$ jobs. Therefore the overall complexity is in $\mathcal{O}(sk+k^4)$. □

If 0 or $p(N)$ are in $\mathcal{F}$, then the same transformation as in Rapine and Brauner (2013) allows to obtain an optimal schedule. Note that, even under a traditional encoding of the instance, the optimal prefix algorithm has a better time complexity than the L-partition algorithm which runs in time $\mathcal{O}(k^3n)$.

3 A polynomial time algorithm for a fixed number of FSE

In this section we establish that the problem $1|\text{FSE}|C_{\max}$ can be solved in polynomial time under a HM encoding of the instances if the number of forbidden instants is fixed, that is, if k is not part of the input. This result extends a theorem from Rapine and Brauner (2013) which establishes that $1|k-\text{FSE}|C_{\max}$ is polynomial under a standard encoding, that is, its complexity is polynomially bounded in n, the number of jobs (but not in s, the number of types). The rest of the section is devoted to proving the following theorem:

Theorem 3 *The problem* $1|\text{FSE}|C_{\max}$ *is Fixed Parameter Tractable for parameter* k*, even under high-multiplicity encoding of the input.*

Notice that if the instance is of large diversity, the optimal prefix algorithm (Algorithm 1, Section 2) can deliver an idle-free (and thus optimal) schedule in time $\mathcal{O}(s)$ for any fixed number k of forbidden instants. Hence we can focus on the case of small diversity instances. Our idea is to formulate the problem on small diversity instances as an integer linear program (ILP) with a fixed number of variables and constraints. Such an ILP can be solved in polynomial time, due to the following result from Eisenbrand (2003):

Theorem 4 (Eisenbrand (2003)) *An integer program of binary encoding length* l *in fixed dimension, which is defined by a fixed number of constraints, can be solved with* $\mathcal{O}(l)$ *arithmetic operations on rational numbers of binary encoding length* $\mathcal{O}(l)$*.*

For small diversity instances, we have by definition $s \leq k$. Thus, if the number of variables and the number of constraints in our ILP formulation are bounded by a polynomial in k, Theorem 4 implies that $1|\text{FSE}|C_{\max}$ is FPT with respect to parameter k. Clearly, to obtain such a formulation, we can not afford to introduce one decision variable (such as the completion time) or one constraint (such as avoiding to complete on a forbidden instant) for each job. Instead, as already discussed in the HM encoding of a solution, see Section 1, we represent a solution by the number of jobs of each type scheduled between two consecutive forbidden instants. However, this representation of the solution is suitable only for idle-free schedules, since otherwise one has also to give the starting time of each block of jobs. To formulate the problem as an ILP, we take advantage of the fact that any large diversity instance admits an idle-free schedule, see Theorem 1. More precisely, we transform a small diversity instance I into a large diversity instance I' by adding dummy jobs as follows. Given an instance I composed of s types, I' is constituted of the following types:

- *Real jobs.* They are the jobs of I. We denote by p_i and m_i the processing time and the multiplicity of the type i, for $i = 1, \ldots, s$.
- *Optional jobs.* We add $k+1$ types to ensure that there exists an idle free schedule. For $i = s+1$ to $s+k+1$, type i has a processing time $p_i = i - s$ and its multiplicity is unbounded.

The number of jobs of the instance I' is unbounded due to the optional jobs. However, as their name suggests, a schedule π' for I' does not need to schedule all the *optional* jobs. More precisely, we do not request to schedule any optional job once all the *real* jobs have been processed and all the forbidden instants have been crossed. We denote by $\widetilde{C}_{\max}(\pi')$ the completion time of the last *real* job of π'. We have the following property:

Property 1 There exists a schedule π for the instance I with makespan $C_{\max}(\pi)$ if and only if there exists an idle-free schedule π' for the instance I' such that $\widetilde{C}_{\max}(\pi') = C_{\max}(\pi)$.

Proof Given a schedule π' for I', we immediately obtain a valid schedule for the instance I by replacing the optional jobs by idle times with the same duration. The jobs of I are processed at the same dates as in π', and thus clearly the makespan is equal to $\widetilde{C}_{\max}(\pi')$. Conversely, consider a schedule π for instance I. We have to prove that for each idle period $[u, v]$ occurring in π, we can sequence optional jobs to obtain an idle-free schedule. Since π is feasible, u and v cannot be forbidden instants. Thus, if the idle period is short, that is

$v - u \leq k + 1$, we can simply schedule an optional job of duration $v - u$. Otherwise, we have $v \geq u + (k + 2)$. Since there are k forbidden instants in the instance, at least one instant in the time interval $[u + 1, u + k + 1]$ is not forbidden. Let t be the last forbidden instant before $u + 1 + k$ which is not forbidden. In π', at time u, we schedule an optional job of duration $t - u \leq k + 1$. By immediate induction we can fill the remaining idle period $[t, v]$ with optional jobs. □

Based on Property 1, we show that we can use an ILP with a fixed number of variables and constraints to find an idle-free schedule π' minimizing the completion time of the last real job. We denote by $s' \geq k + 1$ the number of types (real and optional) in the instance I'. By construction I' is of large diversity, that is $s' > k$, and thus we know that an idle-free schedule π' exists. To bound the completion time of the last real job, we use the property (see Rapine and Brauner (2013)) that any list scheduling algorithm produces a schedule with makespan at most $Q = 2k + \sum_{i=1}^{s} m_i p_i$. Thus Q is an upper bound on the completion time of the last real job in an optimal schedule for I'. As a consequence we can assume without loss of generality that $\gamma_k \leq Q - 2$, since the last FSE instants can be discarded till this inequality holds. We also add a very large optional job of processing time $p_{s+k+2} = \gamma_k + 1$. This job allows to cross all the remaining FSE instants if the schedule finishes before the last one. Finally, for the ease of presentation, we introduce the notation $\gamma_{k+1} = Q + p_{s+k+2} + 1$.

As already discussed, we can represent an idle-free schedule by giving the number of jobs of each type sequenced between any two consecutive forbidden instants (or alternatively by giving the cumulative number of jobs completed before any forbidden instant) and the jobs crossing forbidden instants. We have the following decision variables:

m_{ij}	number of jobs of type i completed by time γ_j for $i = 1, \ldots, s'$ and $j = 1, \ldots, k + 1$.
$S_{jf} = 1$	if a job crosses exactly the instants γ_j till γ_{f-1} (included), for $j = 1, \ldots, k$ and $f = j + 1, \ldots, k + 1$.
$= 0$	otherwise
$x_{ij} = 1$	if a job of type i crosses the instant γ_j and this job does not cross the previous FSE instant, for $i = 1, \ldots, s'$ and $j = 1, \ldots, k$.
$= 0$	otherwise
$y_j = 1$	if all real jobs have been completed by time γ_j, for $j = 1, \ldots, k$.
$= 0$	otherwise
$\widetilde{C}_{\max}$	completion time of the last real job.

The variables m_{ij} are non-negative integers, S_{jf}, x_{ij}, y_j are boolean variables and $\widetilde{C}_{\max}$ is a non-negative real. We also define variable W_j as the total work completed by time γ_j for $j = 1, \ldots, k + 1$. Notice that we do not distinguish real from optional jobs in the definition of W_j, that is W_j is simply a short-hand for $\sum_{i=1}^{s'} p_i m_{ij}$. Also notice that W_j does not take into account the processing time of a job started but not yet completed, that is a job that would cross the forbidden instant γ_j. Hence a job crossing the forbidden instants γ_j but not γ_{j-1} must start at time W_j in an idle-free schedule.

The following linear formulation finds an idle-free schedule minimizing the completion time of the last real job for the instance I':
Minimize $\widetilde{C}_{\max}$, subject to the constraints:

– All FSE are crossed, which is equivalent to require that variables S_{jf} define a $1 - (k + 1)$ path:

$$\sum_{f=2}^{k+1} S_{1f} = 1 \tag{1}$$

$$\sum_{j=1}^{k} S_{j,k+1} = 1 \tag{2}$$

$$\sum_{j=1}^{f-1} S_{jf} = \sum_{l=f+1}^{k+1} S_{fl} \qquad \forall f = 2, \ldots, k \tag{3}$$

– A job crosses γ_j as its first FSE instant if and only if $S_{jf} = 1$ for some index $f > j$:

$$\sum_{i=1}^{s'} x_{ij} = \sum_{f=j+1}^{k+1} S_{jf} \qquad \forall j = 1, \ldots, k \tag{4}$$

– For each type, the variable m_{ij} is increasing with j. In addition, if a job of type i crosses γ_j, then the number of jobs of type i completed should increase by at least one after the next forbidden instant following the completion of the job.

$$m_{i,j+1} \geq m_{ij} \qquad \forall i = 1, \ldots, s' \quad \forall j = 1, \ldots, k \tag{5}$$

$$m_{if} \geq m_{ij} + x_{ij} + S_{jf} - 1 \qquad \forall i = 1, \ldots, s' \quad 1 \leq j < f \leq k + 1 \tag{6}$$

– Schedule all the real jobs:

$$m_{i,k+1} = m_i \qquad \forall i = 1, \ldots, s \tag{7}$$

– Set $y_j = 0$ if all the real jobs are not completed before the instant γ_j:

$$\sum_{i=1}^{s} m_{ij} \geq y_j \sum_{i=1}^{s} m_i \qquad \forall j = 1, \ldots, k \tag{8}$$

– Definition of the work W_j:

$$W_j = \sum_{i=1}^{s'} m_{ij} p_i \qquad \forall j = 1, \ldots, k+1 \tag{9}$$

– All the work W_j must be completed by time γ_j:

$$W_j \leq \gamma_j - 1 \qquad \forall j = 1, \ldots, k+1 \tag{10}$$

– The amount of work completed can not increase between instants γ_j and γ_{f-1} if a job crosses these instants, that is $S_{jf} = 1$:

$$W_{f-1} \leq W_j + Q(1 - S_{jf}) \quad \forall 1 \leq j < f \leq k+1 \tag{11}$$

– If $S_{jf} = 1$ and a job of type i crosses γ_j, then this job should complete in the time interval $[\gamma_{f-1} + 1, \gamma_f - 1]$:

$$W_j + \sum_{i=1}^{s'} p_i x_{ij} \geq \sum_{f=j+1}^{k+1} (\gamma_{f-1} + 1) S_{jf} \qquad \forall j = 1, \ldots, k \tag{12}$$

$$W_j + \sum_{i=1}^{s'} p_i x_{ij} \leq \gamma_j - 1 + \sum_{f=j+1}^{k+1} (\gamma_f - \gamma_j) S_{jf} \qquad \forall j = 1, \ldots, k \tag{13}$$

– The makespan should be equal to the first W_j such that $y_j = 1$:

$$\widetilde{C}_{\max} \geq W_1 \tag{14}$$

$$\widetilde{C}_{\max} \geq W_j - y_{j-1} Q \qquad \forall j = 2, \ldots, k+1 \tag{15}$$

Constraints (1)-(2)-(3) are classical flow conservation equations. They impose all the forbidden instants to be crossed in an idle-free schedule. If a job crosses the forbidden instants γ_j up to γ_{f-1}, Constraint (4) ensures that exactly one variable x_{ij} is set to 1 to represent the type of this job ; Reciprocally if one job crosses γ_j and not the preceding forbidden instant, Constraint (4) ensures that exactly one variable S_{jf} is set to 1, to represent the set of FSE instants crossed by the job. Constraint (6) forces the number of completed jobs of type i to increase by at least one between forbidden instants γ_j and γ_f if a job of type i crosses exactly all the FSE instants from γ_j to γ_{f-1}. Notice that in this case we have $x_{ij} = 1$ and $S_{jf} = 1$, which imposes that $m_{if} > m_{ij}$. As we know that an optimal schedule sequences the last real job before instant γ_{k+1}, we can impose through Constraint (7) that all the real jobs are completed by this time.

Constraint (10) ensures that the completion time of the last job completing before the instant γ_j does not coincide with this instant. Constraint (11) prevents from scheduling some jobs between forbidden instants crossed by a same job: if variable S_{jf} is equal to 1, then the constraint boils down to $W_{f-1} \leq W_j$. Due to Constraint (5), W_l is increasing with the index l. Hence we have $W_j = W_{j+1} = \cdots = W_{f-1}$: The work achieved by time γ_{f-1} is still W_j. On the contrary if S_{jf} is equal to zero, the constraint becomes redundant.

Constraints (12) and (13) prevent a job crossing the forbidden instant γ_j from completing on another forbidden instant. If $S_{jf} = 1$ for some index $f \leq k$ and the crossing job is of type i ($x_{ij} = 1$), the constraints force $\gamma_{f-1} + 1 \leq W_j + p_i \leq \gamma_f - 1$. Notice that if $f = k+1$, Constraint (13) becomes redundant. Finally if $S_{jf} = 0$ for all indices $f > j$, both constraints are redundant since all the variables x_{ij} are zero due to Constraint (4) already discussed, and the right hand sides are then equal respectively to 0 and $\gamma_j - 1$. Thus (12) states that W_j is non negative and (13) gives Constraint (10).

Finally, consider Constraint (15), and let l be the first index such that $y_l = 1$. We claim that this constraint imposes at the optimum that $\widetilde{C}_{\max} = W_l$. Indeed, if $y_{j-1} = 1$ the constraint yields $\widetilde{C}_{\max}$ positivity and if $y_{j-1} = 0$, it boils down to $\widetilde{C}_{\max} \geq W_j$. Setting $y_j = 1$ for all $j \geq l$ is feasible and dominant. Since we are minimizing $\widetilde{C}_{\max}$, the inequality $\widetilde{C}_{\max} \geq W_l$ is tight. We claim that W_j is precisely equal to the completion time of the last real jobs in an optimal solution. Indeed, once this last job has been scheduled, if there are some forbidden instants remaining, they can all be crossed by using the optional job $s+k+2$. This optional job clearly crosses all the remaining forbidden instants, and in particular the instant γ_l. This shows that the value of W_l, and thus of $\widetilde{C}_{\max}$ at the optimum, is equal to the completion time of the last real job.

This integer program delivers an optimal solution to the instance I' and, using Property 1, we can convert it into an optimal solution to the original instance I. Moreover, the number of decision variables of the ILP is in $\mathcal{O}(k^2)$ and the number of constraints is in $\mathcal{O}(k^3)$. Thus, we can apply Theorem 4, which proves Theorem 3.

4 Conclusion

In this paper, we have generalized to high-multiplicity the results from Rapine and Brauner (2013): we have shown that large diversity instances can be solved in polynomial time also with a high-multiplicity encoding of the input. We proposed an algorithm solving this problem in $\mathcal{O}(sk + k^4)$ time, improving the complex-

ity of the previous algorithm from Rapine and Brauner (2013) even if the input is not provided using a compact encoding.

We modeled $1|\text{FSE}|C_{\max}$ as an integer program and used the existence of an idle-free schedule for large diversity instances to avoid modeling the completion time for each job. The resulting integer program has a fixed number of constraints and variables. Therefore, by Eisenbrand's theorem, $1|\text{FSE}|C_{\max}$ is fixed-parameter tractable, even under high-multiplicity encoding of the input. Such an approach could be used on other high-multiplicity scheduling problems to classify them.

Further research can investigate small diversity instances. Especially, it would be interesting to determine whether or not this problem remains polynomial when s is close to k, in particular if $s = k$.

Other optimization criteria such as minimizing the mean flow time can be investigated as well. Chen et al (2013) have already studied a similar problem, with one operator non-availability period. Further investigations of these problems would be interesting and likely to have important industrial applications.

References

Billaut JC, Sourd F (2009) Single machine scheduling with forbidden start times. 4OR 7(1):37–50

Brauner N, Crama Y, Grigoriev A, Van De Klundert J (2005) A framework for the complexity of high-multiplicity scheduling problems. Journal of combinatorial optimization 9(3):313–323

Brauner N, Finke G, Lehoux-Lebacque V, Rapine C, Kellerer H, Potts C, Strusevich V (2009) Operator non-availability periods. 4OR 7(3):239–253

Chen Y, Zhang A, Tan Z (2013) Complexity and approximation of single machine scheduling with an operator non-availability period to minimize total completion time. Information Sciences 251:150–163

Clifford JJ, Posner ME (2001) Parallel machine scheduling with high multiplicity. Mathematical programming 89(3):359–383

Eisenbrand F (2003) Fast integer programming in fixed dimension. In: Battista G, Zwick U (eds) Algorithms - ESA 2003, Lecture Notes in Computer Science, vol 2832, Springer Berlin Heidelberg, pp 196–207

Filippi C, Agnetis A (2005) An asymptotically exact algorithm for the high-multiplicity bin packing problem. Mathematical programming 104(1):21–37

Filippi C, Romanin-Jacur G (2009) Exact and approximate algorithms for high-multiplicity parallel machine scheduling. Journal of Scheduling 12(5):529–541

Hochbaum DS, Shamir R (1991) Strongly polynomial algorithms for the high multiplicity scheduling problem. Operations Research 39(4):648–653

Rapine C, Brauner N (2013) A polynomial time algorithm for makespan minimization on one machine with forbidden start and completion times. Discrete Optimization 10(4):241–250

Rapine C, Brauner N, Finke G, Lebacque V (2012) Single machine scheduling with small operator-non-availability periods. Journal of Scheduling 15(2):127–139